The BopBot

Austin Reilly, CSE, Mathew Cierpial, CSE, Max Jaffe, EE, and Vee Upatising, CSE

***Abstract*—The most memorable songs often feature a compelling (vocal) melody. Musicians often struggle to write melodies that are catchy and unique. The BopBot can inspire this creative process by offering melodies based on what you play into the machine. After inputting a basic chord progression, the BopBot will generate and play back a cohesive and interesting sequence of notes using machine learning.**

I. Introduction

With the introduction of the radio, musicians became able to share their musical creations to a much larger audience. Instead of exclusively sharing their music through physical media and in live performances, music could be spread to anyone with a radio. In doing this, a pressure was applied to musicians to create music that had broad appeal. Catchy music catches the attention of people quickly scanning through radio stations. Although the definition of "catchy" can be hard to define, it is usually indicative of one thing: a memorable melody that gets stuck in listeners heads played on top of a usually simple chord progression.

A. Significance

Where a chord progression is generally quite simple and follows a generic pattern, melodies can be highly complex, fast, and use a higher variety of notes. Fortunately, when a chord is played, a "key" is established which limits the number of melody notes that would sound align well with the current chord. Despite this, melodies can use varying timings and change octaves, making them even more difficult to create. A full understanding of music theory is very helpful in understanding what chord corresponds to what key and timings of notes; however, it takes many years of study to fully understand music theory, and thus everyday musicians often lack this knowledge. Because of this, writing a chart-topping melody can be a very daunting challenge [11].

B. Context and Existing Products

Music generation is a common application of machine learning and is a heavily researched topic [10]. Although music generation using machine learning has been done many times, it has not been done in the same way that the BopBot will do it.

AIVA [8] is a subscription-based artificial intelligence program that generates emotional soundtrack music based on the selected genre or an existing MIDI file. One of the core differences between AIVA and the BopBot is the input and output. AIVA is intended to be used as a tool to replace hiring someone to write compelling themes for the user's projects. Because of this, AIVA limits the user input to a genre or a track of music that it will be "inspired" by. The BopBot, on the other hand, takes a chord progression as an input and outputs a melody that is overlaid on top of the chord progression. Additionally, the BopBot will be functional in a live setting using the MIDI input from a keyboard whereas AIVA cannot take live input.

Google Magenta [9] is an open-source program that uses machine learning to append similar melodies to that which the user inputs. This is different than the BopBot in similar ways to AIVA. Firstly, Google Magenta takes a melody as input where the BopBot takes a chord progression as input. Secondly, Google Magenta is not intended to be used with live input.

C. Societal Impacts

The BopBot is intended to be used by everyday musicians. Because the BopBot will be in the form factor of a music foot pedal, as seen in Table 1, it will be able to be used in conjunction with the equipment and gear that musicians already own. Although the BopBot can be used as a tool to create melodies, it is intended to be used more as a creative-assistance tool. If a musician creates an interesting chord progression, they can use the BopBot to inspire their creative process of writing a melody. The BopBot can also be used as a tool in a jam session. In this way, it can act as another musician with his/her own unique ideas based upon the chord progression being jammed to. A concern of ours regarding the BopBot is that it will be used as a crutch for newer musicians to create melodies. Although the BopBot can be used in this way, it may instill bad habits on learning musicians.

D. Requirements Analysis and Specifications

System Requirements	System Specifications
Sizing	Can fit into a musician's pedal board (typically around 170x138mm).
Power	Powered by a typical music pedal power supply (9V, 1700mA maximum).
Timing	Completes melody generation in an amount of time that is conducive for live music. Less than 5 seconds is acceptable.
Musical Requirement	4 musical genres to choose from for models: Rock, Blues, Classical, Pop.
Machine Learning Performance	Neural network design must have less than 23,000 neurons in each of the 3 layers (fewer if using 4 layers) in order to generate a melody within 5 seconds on a 1 GHz processor.
Memory Usage	Each neural network must be smaller than 200 MB such that the microprocessor has enough data memory (SDRAM).

Table 1: System Requirements and Specifications

The requirements and specifications, seen in Table 1, constrain the problem that the BopBot addresses. As discussed in section C, the BopBot must be in the form factor of a musician's pedal board and must be powered by the standard 9V used in all pedals. In addition, the BopBot must generate the melody within a reasonably quick 5 seconds and be able to create music to several genres. Given this 5 second limit, the BopBot requires quite powerful hardware. Assuming a 1 GHz processor is used, the BopBot's machine learning algorithm must be designed to have fewer than 23,000 neurons in each of its 3 layers. Additionally, each genre's model must be smaller than 200 MB to fit in the processor's data memory.

II. DESIGN

A. Overview

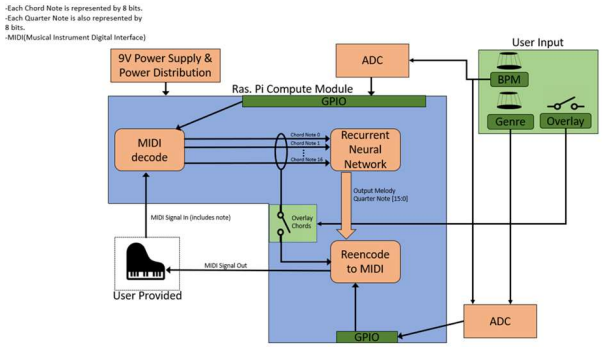

Figure 1: Block Diagram

To solve this problem, we are using a recurrent neural network. We decided to use this type of machine learning network for several reasons which will be explained in detail in the section that goes over block 2. We are using a Raspberry Pi [6] as our development platform and moving to a Raspberry Pi Compute Module [7] for our final PCB. We decided to use the Raspberry Pi because it suits our needs well. We needed a system capable of running an operating system so we could easily compile the code that is necessary to generate our melodies via machine learning. We also needed a fairly powerful processor to ensure that we completed melody generation in a reasonable amount of time (5 seconds).

The first major block in the block diagram, seen above in Figure 1, is the MIDI decode stage. Roughly speaking, this is the "recording stage" where the user input from the keyboard is turned into a format that can be used with our machine learning models to generate a melody. As you can see in the block diagram in Figure 1, the decoded inputs (4 chords) are sent to the next major block in our diagram, the recurrent neural network. After the second block has generated a melody, the output is then sent to the third and final major block in our design, the MIDI encode stage. This can be thought of as the "playback" stage where the output from the machine learning is reformatted into MIDI so it can be output to a speaker. Besides the three main stages, there are several peripherals which allow for the user to alter different settings concerning the recording and playing back parameters.

Specifications played a pivotal role in the performance of the overall system. The timing specification imposed the most tradeoffs in the overall design of the system. Although the machine learning model was improved over the course of the entire design, melody generation was quick, often less than two seconds. The more major timing tradeoff was in loading different models. This would often take 5-10 seconds when switching, for example, from the rock to classical models. Although this did fully satisfy the specification, this was not considered as failing the specification either because users are not intended to change genres often. Musicians typically write a certain genre of music, thus they will not feel the need to switch genres often. Most likely, users would not change genres within sessions of using the BopBot.

B. Block 1 - MIDI Decode

The first stage to happen in our project is the MIDI Decode stage. This stage can be further broken down into two separate steps. The first of which is sampling the user input at the correct time steps. Human beings struggle significantly with keeping consistent time. Even trained drummers, or lifelong professional musicians struggle to keep consistent and accurate time. Below is a plot [1] of the tempo variation for Led Zeppelin's classic song "Stairway to Heaven"

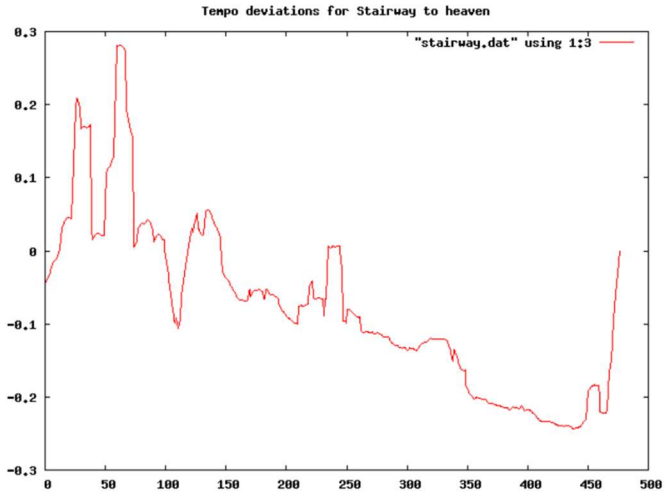

Figure 2: Tempo variations over the course of "Stairway to Heaven" by Led Zeppelin. Measures the average beat variation from the average BPM of the song. Y-axis is variation, X-axis is time.

As you can see from Figure 2, keeping consistent time is something that is virtually impossible for any given person. To fix this issue we implemented something we referred to as a "rounding interval". Due to our machine learning inference architecture, the input into the Recurrent Neural Network is four chords, we sampled the user input at the start of each bar line. A visual representation of the rounding interval is given in Figure 3.

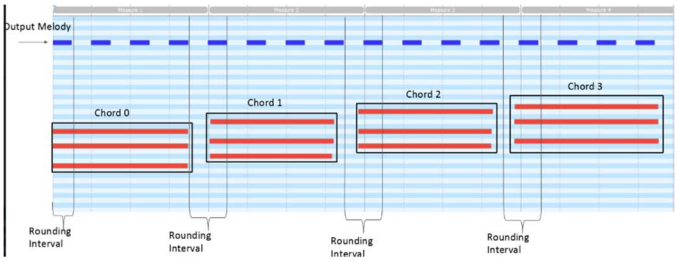

Figure 3: Visual representation of the rounding interval implemented to correct for variations in user input tempo.

To choose a proper amount of time we conducted various trials measuring each group members variation from proper timing. To do this we played a metronome at 120 BPM, in 4/4 time and simply printed out the difference in time between correct time and the time the user played a note. We found that after conducting 10 trials per team member, no variation was greater than 0.25s before or after the bar line. Furthermore, our rounding interval was chosen to be plus/minus 0.25s from the bar line.

Now that a means to account for incorrect timing from human input has been implemented, the MIDI notes would be recorded and saved to be used in the melody generation. In the early stages of the BopBot, algorithms were to be used to decode the chord played at each interval. This would involve reading all notes played during an interval and converting them to a single major or minor chord, represented as a value between -1 and 23. In later iterations of the melody generation software, this was found to be an unnecessary. Instead, the first four notes played during the rounding interval would be saved exactly as they are and sent to be used in machine learning. It was chosen to be the first four notes during the interval because a large majority of chords only use less than four notes. This would increase the complexity of the problem tackled by the machine learning. Now, the output of the MIDI decode stage and input to the neural networks would be an array of sixteen notes, which would be divided into groups of four to represent each chord. If less than four notes were played during the interval, the empty spaces in the array would be automatically filled as rests.

C. Block 2 - Recurrent Neural Network

After the MIDI notes are successfully decoded, the information is then passed to the second block, the Recurrent Neural Network. A Recurrent Neural Network takes input one at a time, preserving the sequential ordering of the data. In this case, a sequence of four chords will be passed in as input to the Recurrent Neural Network. Each value is a summary of a chord mapped to a specific integer value ranged -1 to 23 as detailed in the explanation of Block 1. Once the input has been passed in, the Neural Network performs inference, also known as forward propagation. This generates a sequence of 16 notes that correspond to the four chords that were given as input. The Neural Network generates this sequence of notes one at a time and passes the information forward. This means that every note generated has context of the notes that were previously generated before it. The raw values of the generated sequence of notes are then passed to the third block, MIDI Encode, for necessary preprocessing.

The use of the Recurrent Neural Network was a specific design choice. A Recurrent Neural Network is a very special type of Neural Network that deals with temporal data. This is data that has a sequential ordering as well non-independent instances over time. For this specific problem, every note value is directly dependent on the values of the notes around it. A Recurrent Neural Network is able to capture this information in a way that non-temporal Neural Networks are not be able to.

The second design choice corresponding to this block was the use of LSTM (Long Short Term Memory) layers inside the Recurrent Neural Network. LSTM layers are a special kind of hidden layer within a Recurrent Neural Network that allows the network to remember long-term time dependencies. The first design of this block did not include LSTM layers. After testing the network showed that it lost its context around the 10th note it generated. After which the network generated the exact same value for the rest of the 6 notes. This is known in the Machine Learning field as the Vanishing Gradient Problem. To combat this, the hidden layers of the Recurrent Neural Network were replaced with LSTM layers. This allowed the Neural Network to keep track of long-term dependencies and keep its context all the way up to the 16th note.

The current design of the Neural Network includes 3 LSTM layers with 256 nodes at each layer, which feeds into 3 Dense layers with 512 nodes at each of those layers. There are currently 3 Neural Networks each trained on a different genre of music. The current genres of music are Pop, Blues, and Classical.

Given that this is a Machine Learning problem, there are a multitude of ways in which the Neural Network's performance can be evaluated. The current metric of evaluation we are using is Mean Squared Error. Because this is a supervised learning problem, for every input in the dataset there is also a corresponding output that is considered ideal. During the training process, the Neural Network generates an output for a given input. This output is then compared to the corresponding ideal output and the Mean Squared Error is calculated. This is the Loss Function that is defined for each of the networks.

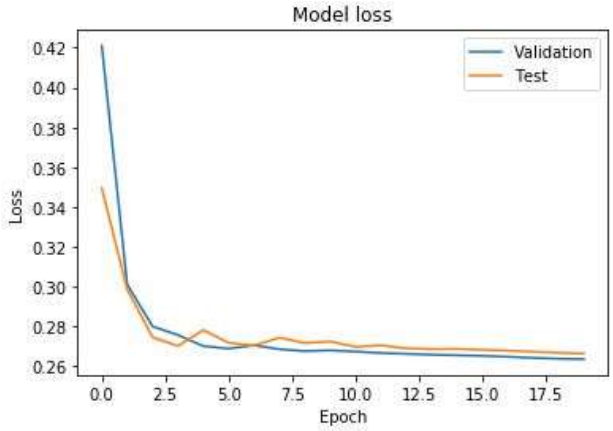

Figure 4: Training and Validation loss of Pop model

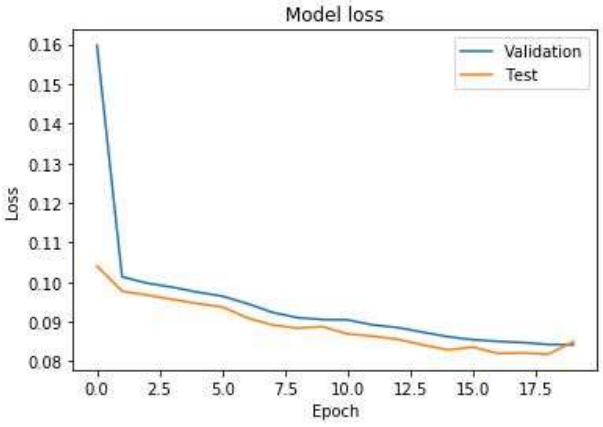

Figure 5: Training and Validation loss of Blues model

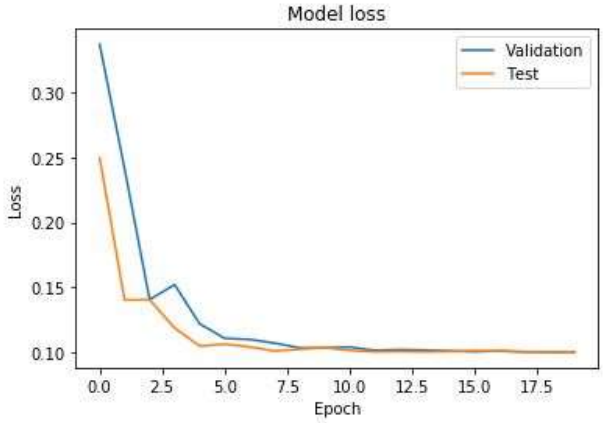

Figure 6: Training and Validation loss of Classical model

The Loss Function represents how inaccurate each Neural Network is. Over time, the Loss Function shows the rate at which each Neural Network is learning from its mistakes. It shows how each model is becoming less inaccurate after every iteration. The three models shown were stopped after 20 training iterations.

At the start of the training process the dataset is divided into three chunks, Training, Testing, And Validation. Training takes 80% of the dataset while Testing and Validation take 10% each. The data inside the Training set is what the Neural Network learns from, and at every iteration the Neural Network uses the data inside the Testing set to calculate the Testing loss. This loss value indicates how close the Neural Network is to reaching an ideal output and is something that the Neural Network works to minimize. The data inside the Validation set is used to calculate the Validation loss. This loss metric is meant to give the most accurate representation of a Neural Network model's performance. This is because the Validation data is never shown to the Neural Network during the training or testing process. The Validation loss is silently calculated in the background at every iteration and gives us a metric of seeing how well the model performs on data that it has never seen before. This is a useful thing to measure because Neural Networks often become overfit to that data that is has been trained on. Becoming overfit means that the Neural Network performs very well on training data but poorly on new data. This generally means that the variance of this model is too high. If the Testing loss goes down while the Validation loss goes up, this is a pretty good indication that the model is becoming overfit to the data that it has been given.

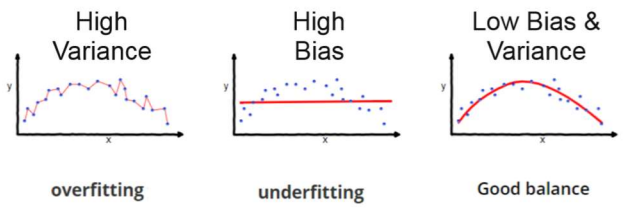

Figure 7: Relationship between overfitting, underfitting, bias and variance

As of right now, the current evaluation metric of Mean Squared Error works fine, but in the future, we have other evaluation metrics we would like to implement. One method we would like to implement is a single blind trial in which each participant listens to a compilation of melodies and tries to classify each melody as either human-composed or AI-generated. The purpose of this trial would be to get a true evaluation metric of whether a generated output is "good". If a generated melody can fool a participant into thinking that it is human-composed, then that output would be evaluated positively and vice versa. This evaluation metric would be a lot more representative of a model's performance rather than the Mean Squared Error between a generated output and the ideal output.

D. Block 3 - MIDI Encode

The last significant block in our design is the MIDI encode block. In this block we take the output from block 2, the machine learning algorithm, and turn it back into a listenable format. This requires post processing of the data. The output of the machine learning algorithm are numbers in a continuous three-dimensional space. Thus, the numbers will almost always be floats and need to be rounded to the nearest integer value which represents the MIDI note value that the machine learning outputted. We also need to overlay the chords that the user inputted on top of the melody that is generated by the machine learning algorithm. To accomplish this, we used a function in the python library we are using to append the tracks on top of each other. We then allow the user to choose whether they want to hear just the melody that was generated, or the melody that was generated as well as the chords they played as input. Prior to making the final track that will be played the optimal octave for the melody must be selected. We decided to simply choose one octave higher than the average octave of the input chord values for now, but in the future, we plan on allowing the user to select what octave to play the melody in so they can find what sounds best to them. Below is a visual representation of octaves [4].

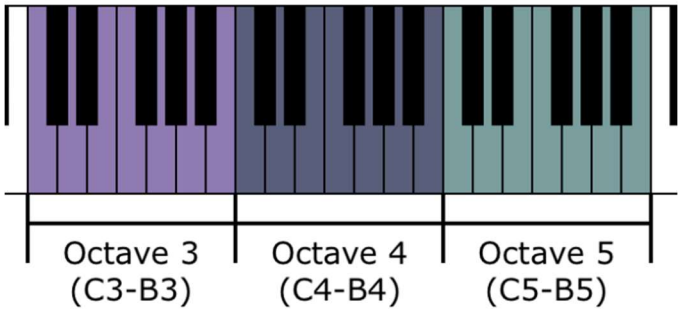

Figure 8: Visual representation of musical octaves.

Each octave has twelve notes in it that correspond to twelve sequential MIDI values. This allows us to easily scale the octave by first normalizing the data to the lowest octave, and then multiplying each note value by twelve times the number of the octave you wish to use. Below is a visual representation of the MIDI note values that demonstrates the idea of octaves well [5].

Octave Number	Notes											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

Figure 9: Visual representation of the musical octaves in MIDI

Besides allowing the user to scale the octave. In the future we will also allow them to choose the tempo at which their generated melody and input chords is played back.

III. The Product

A. Product Overview

The BopBot consists of one independent, totally enclosed system. This system consists of the main electronic hardware, mounted in an aluminum enclosure. The design of the BopBot is intentionally simple in nature, to emulate existing products that musicians know and use on a regular basis. The intent of the design was to be "plug and play" where the musician could have just one piece of hardware, plug their instrument in, and play with very little effort needed. A picture of the design is provided in Figure 10, and the block diagram of the functionality of the design is provided in Figure 1.

Our block diagram is almost entirely software based, as the MIDI decode, Recurrent Neural Network, and Re-Encode to MIDI stages, are all implemented in software (Python) on the Raspberry Pi Compute Module in the enclosure. The user Input sections of the right of the block diagram are physical knobs that are mounted on the face of the enclosure to allow for input parameters such as timing and pitch to allow for variations in generated melodies. The ADC blocks live on the same PCB as the Compute Module within the enclosure and simply provide an interface between the analog potentiometer knobs, and the digital input pins of the Compute Module.

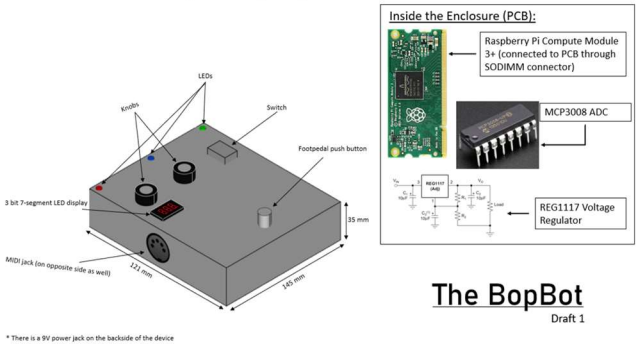

Figure 10: The BopBot Product Sketch

B. Electronic Hardware Component

To design the PCB on which the compute module and ADCs live (among other things), mainly Eagle CAD software was used. The design that was implemented was heavily based on a design specified in [12] and on provided documentation from the Raspberry Pi foundation [2][3]. In addition to the Compute Module and ADC's, the board has a USB- A female connector to provide MIDI output, as well as internet connectivity through an adapter. In the ultimate design the USB-A would not be used for internet connectivity, but in our testing and debugging it was used for this purpose. The PCB also houses voltage regulators to step down the input voltage to the appropriate 5V, 3.3V, 1.1V levels needed by the Compute Module. The board otherwise housed the necessary capacitors, and resistors specified in the Raspberry Pi Compute Module schematics [2], and the Datasheet [3].

C. Product Functionality

At the time of CDR, much of the BopBot was functioning. Although the software and machine learning were improved consistently throughout the process, they were at a presentable state at CDR. The PCB within the BopBot had been delivered and had begun to be populated, but was not yet usable in time for CDR. Instead, the Raspberry Pi Compute Module was used with its corresponding I/O board. After CDR, the PCB was to have a final revision to improve a few issues with the first iteration. In addition, the final revision of the PCB would need to leave space for the ADCs and voltage regulators, which were not yet implemented in the first iteration. Finally, the enclosure was not yet implemented. The box to be used in the final product was purchased, but still needed to have holes drilled into it for inputs and outputs. Switches, buttons, LEDs, and MIDI jacks would then be mounted in these holes.

The demo presented at CDR proved that the BopBot could become a fully functional product. A demo was performed (using the Compute Module and I/O board) at CDR which showed that the machine learning models had reached a point where the generated melodies would sound decent alongside the inputted chords. The Compute Module was set to load the software on boot, and the controls ran flawlessly.

D. Product Performance

To evaluate the performance of the BopBot, please refer to Table 1. Going through this table, each specification was met to the level which we originally set out to accomplish except for the Timing specification. Firstly, the enclosure that was implemented in our final design was a Hammond 1590XX enclosure which is 145mmx121mm, which is slightly smaller than the original sizing that was specified. This specification exists to enforce the simplicity of the ultimate design to suit our target audience. We intended for the design to emulate existing products, so it could be integrated seamlessly into instrument effects boards, and the 145mmx121mm sizing is appropriate for this task. For the power specification, the provided current from a typical music pedal power supply may not have been sufficient, so this specification was likely to be reevaluated given we had enough time. For the timing specification please refer to section 2A which discusses how the timing specification was not met to the original standard. For the musical requirement specification, enough data was collected from each of the genres to create a distinct sound for the generated output, which was the original intent of this specification. The final two specifications were of no concern due to the usage of the Compute Module which has enough

computing power and memory space to accommodate our models, as well as the program which implements functionality.

IV. Conclusion

Currently our project is a Python program on a Raspberry Pi 3 Model B. Our project is currently limited to the major/minor triad inputs outlined in Section II.B. We plan to implement a more robust chord detection algorithm that allows for more types of chords (Inversions, 7's, 9's). We also plan to implement a more robust post-processing program in our MIDI encode stage. Currently the input and output are locked at the same BPM (120) and the octave is chosen for the user. We plan to allow for the tempo to be scaled so that the output can be played faster or slower. We also plan to allow for the user to choose the octave the generated melody is in.

After consulting with our advisor, Professor Moritz, we plan to utilize a Raspberry Pi Compute Module in our larger printed circuit board. We expect this to be fairly involved, but there is extensive documentation from both Raspberry Pi and independent sources. After this is complete, our Python program can be directly ported from the complete Raspberry Pi, to our PCB involving the Compute Module.

Acknowledgment

Team 21 would like to acknowledge our advisor Professor Moritz for his guidance in tackling this problem. We would also like to thank Sachin Bhat for his seemingly endless assistance during our design process.

References

[1] Lamere, Paul. "In Search of the Click Track." *Music Machinery*, musicmachinery.com/2009/03/02/in-search-of-the-click-track/.

[2] "MIDI Files." *Mido 1.2.9 Documentation*, mido.readthedocs.io/en/latest/MIDI_files.html.

[3] "C Major Scale." *Basicmusictheory.com: C Major Scale*, www.basicmusictheory.com/c-major-scale.

[4] "Octave Registers." *All About Music Theory.com*, www.allaboutmusictheory.com/piano-keyboard/octave-registers/.

[5] "Articles." *Noterepeat.com*, www.noterepeat.com/articles/how-to/213-MIDI-basics-common-terms-explained.

[6] *Buy a Raspberry Pi 3 Model B – Raspberry Pi. (n.d.). Retrieved from* https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[7] *Buy a Compute Module 3 – Raspberry Pi. (n.d.). Retrieved from* https://www.raspberrypi.org/products/compute-module-3-plus/

[8] *"AIVA." AIVA,* www.aiva.ai/.

[9] *"Magenta." Magenta, magenta.tensorflow.org/.*

[10] Agarwala, Nipun, Yuki Inoue, and Axel Sly. "Music composition using recurrent neural networks." *CS 224n: Natural Language Processing with Deep Learning,* https://pdfs.semanticscholar.org/c933/79a401dd159fc0c90eab44c43d07286b227e.pdf

[11] Blume, Jason. "The Top 5 Melody Pitfalls-and How to Avoid Them." *BMI.com*, www.bmi.com/news/entry/the_top_5_melody_pitfallsemand_how_to_avoid_them_em

[12] Agkopian, Manolis. "Design Your Own Raspberry Pi Compute Module PCB." *Instructables*, Instructables, 30 June 2019, www.instructables.com/id/Design-Your-Own-Raspberry-Pi-Compute-Module-PCB/.

[13] "Compute Module and Related Schematics." *Compute Module and Related Schematics - Raspberry Pi Documentation*, www.raspberrypi.org/documentation/hardware/computemodule/schematics.md.

[14] *Raspberry Pi Compute Module Datasheet*. Oct. 2016, www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM_1p0.pdf.

Appendix

A. Design Alternatives

We initially planned on completing our MDR design using the De1-SoC development board. We eventually came to realize that it did not serve our purposes well and only served to overcomplicate our design process. This is mostly due to the fact that the De1-SoC has an FPGA that must be configured to gain access to a lot of the components on the board. This added a whole other element to the design process that in the end would not matter for our final design. Instead of the De1-SoC we decided to move to a raspberry pi model 3 B to complete our MDR. Not only is this board extremely easy to develop with, but it also can be used in our final PCB design as a compute module. This will make the porting of our existing software much easier, leaving only the other hardware components to be implemented (Buttons, Switches, USB interface, GPIO pins, ethernet port etc.)

B. Technical Standards

Because the BopBot is intended to be used by musicians, it is important that it can be integrated easily with a musicians standards. Most importantly, the BopBot uses standard MIDI for recording and playback. This is the standard way to read and write from electric keyboards. Because of this, musicians will often own their own MIDI cables that can be used to connect the BopBot to their keyboard. To allow easy integration with other footpedals that musicians may use, the BopBot is designed to use a standard 9V power supply and be of a similar form-factor as other footpedals (see Table 1).

The software used in the BopBot uses a Unix-based operating system, which is an IEEE standard. The operating system is Raspbian which is based on the Linux operating system Debian. Obviously, a Linux operating system is not the exact same thing as basic Unix, but it is very closely related.

C. Testing Methods

The first system specification we were able to test was number two, given in Table 1. To test if our system generated melodies in under five seconds, we simply chose 25 unique chord progression inputs and measured the time it took to generate a melody. The data for the melody generation time was then compiled into Figure 14.

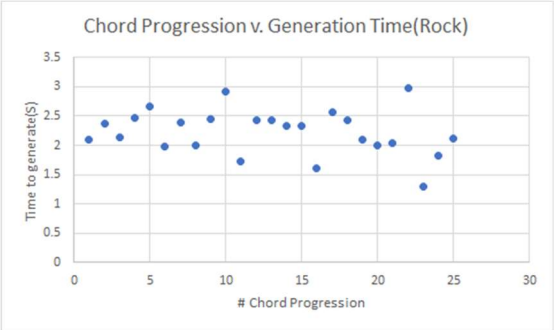

Figure 14: Chord progression inputs versus melody generation time for the rock model

A concern we emphasized at both PDR and MDR, was that our Machine Learning model would not generate melodies that sounded "good." Music is subjective, so defining a metric that determines what is good or bad is nearly impossible. Our initial idea is to simply poll people and ask them if the generated melodies sound good or not. We would have to mix in human-made melodies so that comparisons could be drawn about whether our model generated a good melody, but also a melody that sounds human-made.

D. Team Organization

Max is the project manager and he generally keeps all of us on track in terms of completing work by certain deadlines. Of course, we all have a hand in making sure things are running smoothly so we all give input from time to time and all have our individual technical work to complete.

Our team is working well together. Our individual expertises complement each other well for this project. Vee is the machine learning guru, having taken classes in the subject and conducted personal research. Austin has an interest in hardware design which has been helpful when interfacing with hardware peripherals (buttons, USB interface, GPIO pins, etc.). Matt is most interested in software which has been helpful in implementing our state machine and other code in python. Max has a good musical background which has been very important in bridging the worlds of machine learning and musical creation together.

There has not been a clear moment when a team member served as a leader. Rather we all give our feedback when pertinent and generally the whole project is directed by Max. An example of when Max helped another team member is when we came up with how our machine learning model would be constructed. He helped us decide what made sense in terms of the input and output of the model (structure of input chords and output melodies). Austin has helped Matt out extensively in creating the python code for the program, specifically in interfacing with hardware peripherals like previously mentioned. Vee is always there to discuss the machine learning model to make sure it is structured in a way that makes sense for our purposes.

We had a communication breakdown with our advisor Prof. Moritz when we were trying to use the De1-SoC for MDR. We as a team did not communicate well enough the status of how things were going with using the board which created a situation where we decided very late in the semester to switch to a Raspberry Pi. Once we established that we were not communicating effectively with our advisor we made a much more consistent effort to stay in touch with the progress of our project. This has made life a lot easier because we address problems proactively rather than waiting until there is an issue.

E. Beyond the Classroom

Matt: In order to write the code that implemented the MIDI encode block in our block diagram I needed to develop my python coding skills. I had only used python very briefly once before this project, so I needed to familiarize myself with python syntax and how to import libraries. Besides that, I needed to gain a basic understanding of the machine learning algorithm we utilized, just so I could give input when necessary.

Austin: Like Matt, I implemented the MIDI decode stage in Python, which I had very little experience in. Familiarizing myself with a new programming language, specifically Python, will translate very well to the professional world. Also, in the future, I would like to work in some sort of a hardware or embedded development position, so our work in the upcoming semester I feel will be very helpful in teaching me skills I may need as a professional.

Vee: I needed to do a lot of research in order to implement the Recurrent Neural Network. Before this project I had worked with other types of Neural Networks, but I had never specifically worked with Recurrent Neural Networks and temporal data. Even after I implemented the first design of the Neural Network there was still much research to be done on how to optimize the performance of the Neural Network. I consulted various research papers and the book *Machine Learning Mastery* quite a bit. I went through dozens of designs before ending up with the current design, which is still being optimized. Although the hardest skill I had to learn was how to organize and structure the data into a shape that was required for the Neural Network. I needed to find a way to mathematically represent music in three dimensions of a continuous space. It was challenging to come up with the data representation in the first place but working with the data representation after that was easy. I just needed to write a python script to extract notes information from a MIDI file and store the data in the exact three-dimensional form it needed to be for the Neural Network. I believe that the skills I developed from this data management process will greatly help me in my career in the Data Science field.

Max: Although this project is somewhat out of my interests in the field of microwaves, I have been able to apply what knowledge I have and have learned a lot. My musical knowledge has been very helpful in this music-oriented project. With this knowledge, I have been able to work with Vee to fine tune the machine learning models to be more based in music theory. This project has also helped me expand my knowledge of embedded programming when collaborating with Austin, Matt, and Vee to make sure the blocks of the block diagram interact with each other correctly. I have also gained a great deal

of knowledge in team management and systems engineering. Because I am the project manager, I have been learning how to communicate with people effectively and positively so that work is accomplished in the most efficient way possible.